Problem 1.a. Use the following algorithm.
1. Assume the integers are all positive. If not, do following steps for negated negative integers, and reverse its order.
2. Use counting sort to separate the integers by its length.
3. Then, use radix sort for each groups that separated by step 2.

Correctness of this algorithm is obvious, since that the long length number is always bigger than short length number.

Time complexity of the algorithm is $O(n)$
(1) Let the number of integers is m, and m_i is the number of integers which its length is i.
(2) Then, step 1 takes $O(m) \leq O(n)$
(3) Step 2 takes $O(m+n) = O(n)$
(4) Step 3 takes $\Sigma_i O(m_i \times i) = O(n)$

Problem 1.b. Use the following algorithm.
1. Use counting sort for the first letter of the words.
2. Recursively use counting sort for the next letter of the words for each group which made by previous step. If there are no next letter for some words, use empty string character ϵ which $\epsilon < a < b < \cdots$.

Correctness of this algorithm is also obvious, since it obeys the lexicographic order.

Time complexity of the algorithm is $O(n)$
(1) Let the number of words is m, and l_i is the length of the ith word.
(2) Obviously, step 1 takes $O(m) \leq O(n)$.
(2) Then, ith word could be sorted by recursive step 2 with $\mathrm{O}(l_i)$ times.
(3) Since counting sort requires linear time, step 2 runs in $O(\Sigma_i O(l_i)) = O(n)$.

Problem 2.a. For groups of 3, the recurrence relation is the following: $T(n) = T(n/3) + T(2n/3) + \Theta(n)$, since at least $n/3$ elements are smaller or bigger than the median.
By substitution $T(n) \geq cn \lg n$, $T(n) \geq \frac{cn \lg n}{3} + \frac{2cn \lg n}{3} + \Theta(n) \geq cn \lg n + \Theta(n) \geq cn \lg n$, so $T(n) = \Theta(n \lg n)$.

However, for groups of 7, the recurrence relation is the following: $T(n) = T(n/7) + T(5n/7) + \Theta(n)$, since at least $2n/7$ elements are smaller or bigger than the median.
By substitution $T(n) \leq cn$, $T(n) \leq \frac{cn}{7} + \frac{5cn}{7} + \Theta(n) = cn - (\frac{cn}{7} - \Theta(n)) \leq cn$, so $T(n) = \Theta(n)$.

Problem 2.b. Use the **SELECT** algorithm.
(1) Pick the median for every iteration using **SELECT** algorithm.
(2) Partition the array using the pivot as the median.
(3) Then, the length of left and right partition will $n/2$ (or $n/2+1$).

Time complexity of **SELECT** and partition are both $O(n)$, so the recurrence relation will be $T(n) = 2T(n/2) + O(n)$, so $T(n) = O(n \lg n)$.

Problem 2.c. Use the following algorithm.
(1) If the number of wells are odd, pick the median value of their y coordinates.
(2) If even, choose any point in the two median values of their y coordinates.

Time complexity of the algorithm is trivially linear, since it can be done by **SELECT**.

The algorithm is correct. Let $y_1 \leq y_2 \leq \cdots \leq y_n$ are y-coordinates of wells.
(1) Total distance $d(p) = \Sigma_{y_i>p}(y_i - p) + \Sigma_{y_i<p}(p - y_i)$ when p is y-coordinate of central pipe.
(2) When n is odd, $d(y_m) = \Sigma_{i>m} y_i - \Sigma_{i<m} y_i$ where $m = \lfloor n/2 + 1 \rfloor$.
(3) Consider $p' = y_m + c$ such that $c > 0$. Then, $d(p) = \Sigma_{y_i>p}(y_i - p) + \Sigma_{y_i<p}(p - y_i) = \Sigma_{i \geq m'} y_i - \Sigma_{i<m'} y_i + (m' - m)p > d(y_m)$ where $y_{m'} \geq p \geq y_{m'-1}$.
(4) By symmetricity, $d(p') > d(y_m)$ when $c < 0$. $\therefore d(p)$ is minimum when p is the median.
(5) When n is even, merge two median points($y_{m_1} \leq y_{m_2}$) into their midpoint.
(6) Applying (3)-(4) conclude that d is minimum when p is in the midpoint.
(7) Adding back to the merged point increase d value as only $y_{m_2} - y_{m_1}$ when $y_{m_1} \leq p \leq y_{m_2}$.

Problem 3. Let A is the array; make the recurrence equation as follows.
Let $N[i]$ as the length of longest subsequence that ends at i. Then, $N[1] = 1$, $N[i] = max\{N[j] + 1\}$ for j s.t $(j < i) \wedge (A[i] \leq A[j])$.
Therefore, the following algorithm can find longest monotonic decreasing sequence in $O(n^2)$.

```
1: procedure LDS(A)
2:     N := array(1, 0, 0, ... 0)
3:     P := array(0, 0, 0, ... 0)
4:     max := 0; end := 0
5:     for j = 1 to A.length do
6:         N[i] := 1; P[i] := 0
7:         for j = 1 to i − 1 do
8:             if N[j] + 1 > N[i] AND A[i] ≤ A[j] then
9:                 N[i] := N[j] + 1; P[i] := j
10:            end if
11:        end for
12:        if N[i] > max then
13:            max := N[i]; end := i
14:        end if
15:    end for
16:    SEQ := list()
17:    while end ≠ 0 do
18:        SEQ.appendFirst(end); end := P[end]
19:    end while
20:    return SEQ
21: end procedure
```